SDS: A Framework for Scientific Data Services

Bin Dong*, Surendra Byna*, and Kesheng Wu*

*Lawrence Berkeley National Laboratory, USA. Email: {DBin, SByna, KWu}@lbl.gov

***Abstract*—Large-scale scientific applications typically write their data to parallel file systems with organizations designed to achieve fast write speeds. Analysis tasks frequently read the data in a pattern that is different from the write pattern, and therefore experience poor I/O performance. In this paper, we introduce a prototype framework for bridging the performance gap between write and read stages of data access from parallel file systems. We call this framework Scientific Data Services, or SDS for short. This initial implementation of SDS focuses on reorganizing previously written files into data layouts that benefit read patterns, and transparently directs read calls to the reorganized data. SDS follows a client-server architecture. The SDS Server manages partial or full replicas of reorganized datasets and serves SDS Clients' requests for data. The current version of the SDS client library supports HDF5 programming interface for reading data. The client library intercepts HDF5 calls and transparently redirects them to the reorganized data. The SDS client library also provides a querying interface for reading part of the data based on user-specified selective criteria. We describe the design and implementation of the SDS client-server architecture, and evaluate the response time of the SDS Server and the performance benefits of SDS.**

I. Introduction

Large scientific simulations and experiments typically store data to file systems with a layout that gives the best performance for writing. Research efforts are underway to write data to parallel file systems, such as Lustre and GPFS, with the goal of achieving peak I/O bandwidth [3], [1]. The stored data is either read directly from the file systems it was written to or transferred to file systems on a different machine for analysis. If the data is laid out on the file system in a pattern that is not amenable to the read patterns of the analysis tasks, the read performance will be poor. For example, reading non-contiguous data from a large array achieves poor performance compared to reading contiguous data.

Difference in read and write access patterns also exists in other applications such as database systems. Relational database management systems improve the read performance through techniques such as using indexes, caching frequently accessed data, and storing materialized views. These optimizations are invisible to database users. However, with scientific data, once the data is stored to a file system, the data becomes immutable, and users have to implement any read optimizations explicitly by themselves. The aim of this work is to bring the automatic data management features from database community to scientific data stored in files [14].

Reorganizing data is a proven strategy for improving read performance. Methods such as elastic data reorganization (EDO) [11], 2-D layout [10], and multi-dimensional chunking [9], demonstrate that reorganization of the data according to specific data access patterns improves performance. Our recent study also shows that accessing sorted or transposed data layout speeds up reading data by more than 50X [6].

Despite many efforts showing benefits of reorganizing data, there is a need for automating the process of reorganizing and managing the replicas. We identify three requirements in performing data reorganization automatically: to determine an optimal layout for improving read requests, to perform reorganization of the data, and to read the reorganized data automatically. To ensure ease-of-use, we also propose to enable these features within a familiar high-level I/O interface.

Our vision is to develop a scientific data management system that hides the complexity of achieving peak I/O bandwidth in both the writing and reading phases of data. In this paper, we describe our first step towards that overarching goal: the design and an initial implementation of the Scientific Data Services (SDS) framework. We analyze the requirements involved in the automatic data reorganization process and identify the key requirements for implementing a data reorganization system. We then present our initial implementation of SDS, which is a new and lightweight framework for reorganizing files stored on parallel file systems. Specifically, SDS is able to automatically identify the files that need to be reorganized and to invoke a appropriate data reorganization algorithm (such as sorting and/or transposing the data), and reorganize the files accordingly.

This rest of the paper is organized as follows: we review the requirements for the automatic reorganization of data in Section II. In Section III, we describe the design and implementation details of the SDS framework. Section IV discusses the experimental set up for evaluating the SDS framework and Section V presents the measurements of overhead of using SDS and the resulting read performance. Section VI reviews related work and Section VII concludes the paper with a brief discussion of future work.

II. Requirements of Automatic Reorganization

1) Finding an Optimal Organization: Determining an optimal organization of data for accelerating read operations has been explored by several research activities [10]. A typical strategy is to identify data read patterns of analysis applications, and then determine an organization that improves the locality and parallelism of the data access from the file system. The reorganized data may include one or more copies of the original data. To minimize the storage space requirement, it is important to replicate only the most useful information. For

example, if the analysis operations only need a part of a data set, then we should avoid replicating the whole data set. In this paper, we assume the read patterns and their corresponding optimal organizations are known. We focus on providing a mechanism for performing a selected reorganization and for directing read calls to the reorganized datasets.

2) Performing a Selected Reorganization: Given an efficient organization of data for future read requests, and reorganizing the data automatically, involves a few hurdles. First, an automatic reorganization system needs permission to read the original data and protect the reorganized data. Scientific data is often produced by a user or a group of users and sharing of that data is limited to the owners. A reorganization system needs permissions to read the original data and reorganize it. Second, the system has to read the data and perform the reorganization, such as sorting and transposition, and write the data to the file system efficiently. This step requires computing power and memory resources. To perform this automatically on machines that run jobs in a batch mode, a reorganization system needs to submit batch jobs automatically. Third, after a reorganization job is finished, the system needs to manage the reorganized data and its associated metadata for future read requests. The replicated data also needs to be protected by giving permissions only to the owners of the data. We will describe the implementation for performing data reorganization with our SDS framework in Section III.

3) Reading Reorganized Data Transparently: Assuming there are multiple replicas of data in different layouts on the file system, we need to redirect the read requests to a replica that gives the best read performance. This task has three challenges: selecting a replica of data to achieve the best performance for a read pattern, intercepting a read call, and redirecting the call to the selected data. Finding the best performing data organization is similar to the requirement mentioned above in Section II-1.

A reorganization system needs to recognize the pattern of one or a set of read calls at runtime and match the logical pattern with physical layout that will provide the best performance. This is a challenge we will be addressing in our future work. This paper focuses on solving the second and third challenges: intercepting read calls and directing them to read the data layout that will provide the best performance.

III. Scientific Data Services

In this section, we provide an overview of the Scientific Data Service (SDS) framework. Current implementation of the framework addresses performing a reorganization method and reading reorganized data. We will explain the requirements we have addressed so far in the discussion of the main components of SDS.

A. SDS Framework Overview

Figure 1 shows a high-level overview of the SDS framework. SDS has two main components: the SDS Server and the SDS Client.

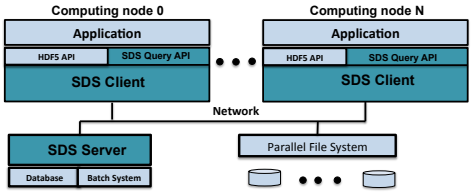

Fig. 1. An Overview of SDS Framework

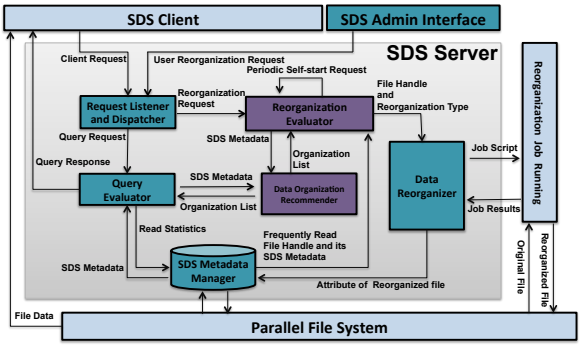

Fig. 2. Design of the SDS Server and interactions among various components. Reorganization Evaluator and Data Organization Recommender are implemented partially and the other highlighted components are implemented fully.

The SDS Server performs reorganization of the data, manages the metadata of the reorganized datasets, and handles the requests from SDS Clients to identify the best version of reorganized dataset to use. The Server handles multiple SDS Clients related to different applications concurrently.

The SDS Client is a light-weight library responsible for intercepting data read calls and for contacting the SDS Server for the location of data to be read. MPI (Message Passing Interface) application processes linked with the SDS Client library read the data directly from the parallel file system and perform any mapping needed between the reorganized data and the read request. Our current implementation supports reading HDF5 data, which is a popular data format used by numerous scientific applications for reading and writing multi-dimensional array data. In addition to HDF5 read calls, the SDS Client also supports range queries with the SDS Query interface. With this interface, applications can request to access data with an SQL like query, such as "`var1 > value1 and var2 <= value2`".

B. SDS Server Design and Implementation

Figure 2 shows an overview diagram of the SDS Server. The server contains the following components: Request Listener and Dispatcher, Query Evaluator, Reorganization Evaluator, Data Organization Recommender, Data Organizer, SDS Admin Interface, and SDS Metadata Manager. We will explain next each of these components in detail.

- The *Request Listener and Dispatcher* handles requests from SDS Clients and the SDS Admin Interface. The clients send metadata requests to verify whether there are any reorganized data files related to a data read

request and to obtain metadata of the reorganized data files, such as the reorganized data file name, offsets, sizes of the data to be read, etc. We developed an SDS Admin Interface for issuing data reorganization requests. Ideally, the SDS Server should be capable of making intelligent decisions on finding data organizations based on known read patterns. However, at this initial stage, we use the Admin interface for providing well-known data organization decisions to the Server. We implemented the communication between the client and the server using protocol buffers or protobuf [12], a message interchange format provided by Google. The client or the Admin Interface send requests comprising file name, dataset name, query or coordinates of the array variable, and type of the request (read, query, or reorganization) encoded into a protobuf message. The Request Listener decodes the message and based on the type of request it dispatches the request either to the Query Evaluator or to the Reorganization Evaluator.

- The *Query Evaluator* analyzes the metadata requests from SDS Clients and performs a lookup for existing reorganized datasets in the *SDS Metadata Manager*. The Metadata Manager maintains relationships between the original data files and their reorganized datasets. Typical SDS metadata includes the name, file location, group name, dataset name for both the original and the reorganized files, and the permissions of the datasets. To ensure the security of the reorganized file, the original file's access permissions for both user id and group id are stored in the SDS metadata. We also store the read statistics for each file in the SDS metadata for tracking the frequency of each file being read. We implemented the SDS Metadata Manager using Berkeley DB, and the Query Evaluator using thread pool. The Query Evaluator is capable of supporting numerous SDS Clients simultaneously. Each thread handles one client request for looking up reorganized datasets, verifying permissions, and sending the location of reorganized file to the client. The Query Evaluator encodes the metadata using protobuf and sends that information to the client.
- The *Reorganization Evaluator* is responsible for deciding whether to reorganize data based on the frequency of access to a dataset and for handling reorganization requests that come from the SDS Admin Interface. It periodically checks the SDS metadata for the most frequently accessed data files and their read patterns for deciding the need for reorganization. The Reorganization Evaluator also serves the *Data Organization Recommender* in selecting the best reorganization data file when multiple reorganized files are available. The current implementation supports the requests from the SDS Admin Interface.
- Based on known read patterns and the original layout of the data on the file system, the *Data Organization Recommender* selects the optimal layout and suggests the Reorganization Evaluator to perform reorganization. This component evaluates the logical view of the read requests and physical data layout of the data for satisfying the read requests, and estimates the latency with various possible physical layouts. We are currently developing a model for choosing optimal data layout.
- The *Data Reorganizer* performs the selected reorganization task. It initiates a reorganization method by generating a batch script to run on a large computer system. We currently support two types of reorganization methods: Parallel sorting and transposition of data. We have shown the benefits of these two data organization methods for two scientific datasets [6]. The SDS Admin Interface is designed to support other reorganization functions to be added in the future. The Data Reorganizer also monitors the health of a reorganization script and after executing the script, it stores the reorganization related metadata into the SDS Metadata Manager database.

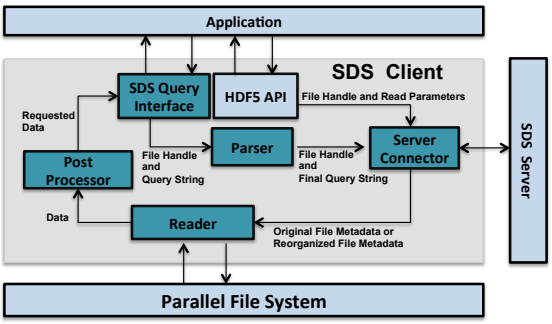

Fig. 3. An overview of the SDS Client and the interactions among various components

C. SDS Client design and implementation

Figure 3 shows an overview of the SDS Client design. The Client supports two types of read interfaces: The HDF5 Read API and SDS Query Interface. The Parser parses the data query request conditions of the Query Interface and the Server Connector communicates with the SDS Server. We implemented the current SDS Client library using the Virtual Object Layer (VOL) of the HDF5 implementation [4]. The VOL supports a mechanism for intercepting HDF5 calls. Our recent paper [6] provides details of our VOL plugin implementation.

- *SDS Query Interface*. The SDS Query Interface provides users the ability to run SQL-style queries on arrays. We extend the HDF5 Read API to accommodate the querying conditions on the specified dataset. The Query API supports returning the count of the data elements satisfying a condition and returning the results of the query to the application.
- The *Parser* verifies query requests for validity, such as whether the requested data file and dataset exists. The Parser passes the attributes of the query conditions to the SDS Server to find the location of a file sorted by the requested dataset. In our current implementation, the execution of the SDS Query depends on whether a dataset

being queried is sorted or not. Supporting queries with the help of bitmap indexes is under development.

- The *Server Connector* packages either a query or a HDF5 Read request into a protobuf data structure and sends it to the SDS Server to retrieve information including the location of the data to be read. In a parallel application using MPI, the SDS Client with MPI Rank 0 sends the request to the SDS server. This strategy avoids all processes of the application asking for the same information from the SDS Server. After the Server Connector on Rank 0 has received the metadata information from the server, it broadcasts the information to all other MPI processes of the application.
- The *Reader* issues HDF5 Read calls based on the metadata information on the location of the dataset. We implemented this as part of the HDF5 VOL with native HDF5 plugin, where HDF5 data is read from the parallel file system.
- When reorganizations such as compression or transposition to a different dimension are performed to achieve better performance, the reorganized data needs to be decompressed or transposed back when the data is returned to the application [6]. Based on the reorganization type, the *Post Processor* performs transformations to present the data in a way that the application expects.

IV. System Configuration

We have deployed our initial implementation of SDS framework on a Cray XE6 supercomputer at the National Energy Research Scientific Computing Center (NERSC), named Hopper [1]. The system has $6,384$ compute nodes, with two 12-core AMD 'MagnyCours' 2.1 GHz processors and at least 32 GB memory per node. We used a Lustre file system, exported as directory */scratch2*, for storing and reading data. We have set the stripe size of the data on the Luster file system to 1MB and the stripe count to 144.

To have a static IP address and port number to serve the SDS Client requests, we ran the SDS Server daemon on a Node Manager (MOM) [2] of the Hopper system. The SDS Server submitted reorganization job scripts to run on Hopper. Using the SDS Admin Interface, we manually sent reorganization requests.

V. Results

In this section, we evaluate the response time of the SDS Server in handling concurrent SDS Client requests and the performance benefit in reading data using the SDS Client library.

A. SDS Metadata Read Response Time

In our current deployment on Hopper, we used one SDS Server. We measured the server response time with the number of SDS Clients varying between 40 and 320 that were requesting for metadata from the Server. We also measured the time for opening and closing HDF5 files with the same number of clients. In both cases, each SDS Client accesses a different file. The SDS Server reads the requested metadata related to a dataset from a database managed by Berkeley DB and returns it to SDS Clients.

Figure 4 compares the response time of SDS with the time spent for HDF5 Open and Close operations by the same number of clients. The x-axis shows the number of clients and the y-axis shows the time in seconds. Each data point refers to the average response time with a range bar showing minimum and maximum response times. The maximum response time of SDS is less than 0.1 seconds in handling up to 200 clients and increases to ≈0.5 seconds in responding to 320 clients. In the same figure, we also show the time needed to perform HDF5 Open and Close operations. It is easy to see that the time needed for SDS to look up the metadata is considerably smaller. On Hopper, the average number of concurrent applications is ≈250. In a production deployment, we expect the number of applications using SDS to be close to a few 10's and therefore the overhead will be negligible.

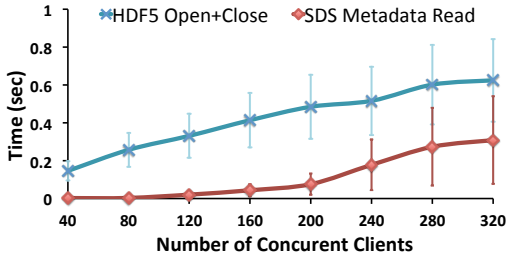

Fig. 4. SDS Metadata Read Overhead from SDS Server

B. Performance of Reading Reorganized Data

To show the effectiveness of using the SDS Client in reading a reorganized dataset, we performed range queries on a large plasma physics dataset. In [6], we have shown that sorting and querying this dataset gives up to 50X performance benefit compared to the traditional way of reading and sifting through the whole dataset for a given query condition. In this paper, we use the new SDS Client and Server implementation to verify that we are achieving similar performance benefits.

In these tests, we use a 2.8TB plasma particle dataset generated by our VPIC simulation [3]. The data contains seven variables: Energy, X, Y, Z, $U_{\parallel}$, $U_{\perp,1}$, and $U_{\perp,2}$. The properties of all the particles is written into a HDF5 file based on X, Y, and Z location. We used the SDS Admin Interface for sorting this dataset based on Energy values using our parallel sorting algorithm explained in [6]. We ran multiple queries on this data with different ranges of Energy values to retrieve particles with a certain Energy threshold, which is a typical query for analyzing this data. The traditional approach of searching for energetic particles is a "Full Scan", where an analysis application reads all the data variables into memory and then selects those where particle energy satisfies a given

[1] http://www.nersc.gov/systems/hopper-cray-xe6/

[2] https://www.nersc.gov/users/computational-systems/hopper/configuration/support-nodes/

TABLE I
SIZE OF THE DATA EXTRACTED BY EACH QUERY FROM A 2.8TB VPIC DATA SET

	$E > 1.10$	$E > 1.15$	$E > 1.20$	$E > 1.25$	$E > 1.30$	$E > 1.35$	$E > 1.40$	$E > 1.45$	$E > 1.50$
Size (GB)	2213.94	904.63	319.85	131.31	63.31	35.00	21.43	13.96	9.35
Percentage(%)	78.67	32.14	11.36	4.66	2.24	1.24	0.76	0.49	0.33

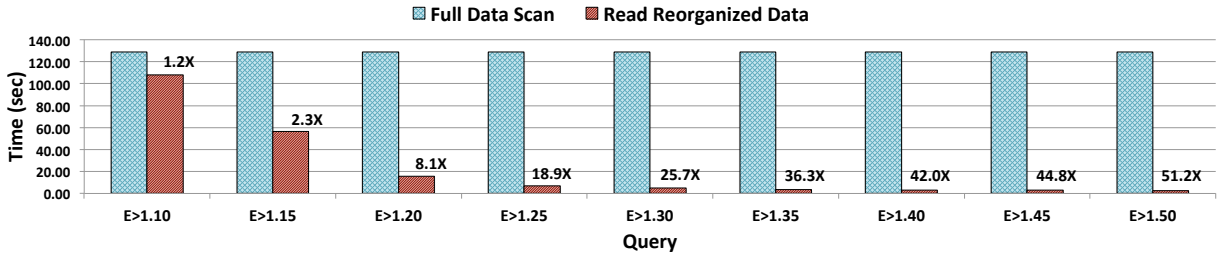

Fig. 5. Reorganized file performance benefit on a 2.8TB VPIC dataset

condition. With the SDS Query interface, we can specify the query condition and read the sorted file only where the energy value condition applies. Table I shows the query conditions, the size of data that satisfies the condition, and the percentage of data to be read from the total dataset. For a query of `E > 1.1`, ≈78% of the data needs to be read and for `E > 1.5`, ≈0.3% of the data is accessed.

Figure 5 shows the performance benefit of querying data with SDS, where the SDS Client requests the Server for the location of an optimal layout of the original data, and reads from a sorted dataset. The full scan of data takes the same time for all cases, as the main cost is reading the data from disk and searching for the data that satisfies a given condition takes a negligible time. In comparison, SDS reads a fraction of the data that satisfies the condition. As the amount of data to be read becomes smaller, the performance benefit increases. We observed 20X to 50X speed up with SDS compared to the traditional full scan method when ≈5% or less of the data satisfies the query condition.

VI. RELATED WORK

Matching the file organization to the data access pattern can improve the data access performance. For a known access pattern, one could usually define a custom file organization. These efforts can be performed at system-level and at file-level. Typically, through sophisticated file allocation strategy, the system-level method minimizes variance of I/O servers [8], improve load balance [15], [16], and reduce I/O contention [7]. The granularity of file-level organization includes striping, disk blocks and so on. Typical file-level organizations include EDO [11], 1DV[10], 1DH [10], 2-D layout [10] and multidimensional chunks [9]. In most cases, these are static methods, where the file organization is determined when they are written to the file systems.

Recently, the ability to query scientific data similar to that of database management systems became an active research field. One idea is to load scientific data into special databases, and then apply a specially designed query language to find data of interest. An example of such an approach is SciDB [2], which provides query and functional languages for querying the data. In most cases, loading scientific data into SciDB is a laborious and time consuming effort. Furthermore, scientists wish to have access to their preferred file format, such as HDF5 or NetCDF. Another set of efforts is focused on building a querying system directly on the raw scientific data files. Typical examples include FastQuery [5], [3], an array-based querying library based on FastBit [13] and FlexQuery [17]. These efforts do not perform reorganization of data based on patterns.

VII. CONCLUSIONS AND FUTURE WORK

Scientific data once written to file systems with a certain data layout becomes immutable. The read performance of subsequent analysis tasks is poor when the read patterns differ from the write patterns. In this paper, we discuss the design and implementation of an automatic data optimization framework for reorganizing and augmenting the original data. We have shown the performance overhead of our framework is minimal and the benefit is in the range of 2X to 50X.

We are exploring several aspects of improving the initial implementation of the framework. As mentioned in Section II, determining an optimal organization of data automatically by matching the logical data access patterns with physical layout of the data is underway. Replication of the full dataset is a typical practice in Hadoop based systems. However, replicating petabytes of scientific data is impractical due to storage limitations. We are exploring methods to limit the replication to frequently accessed data. We are also expanding querying to complex conditions which can benefit for indexing. We plan to use bitmap indexes computed by FastBit technology [13].

ACKNOWLEDGMENT

This work is supported in part by the Director, Office of Laboratory Policy and Infrastructure Management of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and used resources of The National Energy Research Scientific Computing Center (NERSC). We thank

Quincey Koziol and Mohamad Chaarawi from The HDF5 Group for their support with HDF5 VOL.

REFERENCES

[1] B. Behzad, L. Huong Vu Thanh, J. Huchette, S. Byna, Prabhat, R. Aydt, Q. Koziol, and M. Snir. Taming Parallel I/O Complexity with Auto-Tuning. In *Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2013)*, 2013.

[2] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 963–968. ACM, 2010.

[3] S. Byna, J. Chou, O. Rübel, Prabhat, H. Karimabadi, W. S. Daughton, V. Roytershteyn, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin, A. Shoshani, A. Uselton, and K. Wu. Parallel I/O, analysis, and visualization of a trillion particle simulation. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 59:1–59:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[4] M. Chaarawi and Q. Koziol. HDF5 Virtual Object Layer. Technical report, Available: https://confluence.hdfgroup.uiuc.edu/display-/VOL/Virtual+Object+Layer, 2011.

[5] J. Chou, K. Wu, and Prabhat. FastQuery: A Parallel Indexing System for Scientific Data. In *CLUSTER*, pages 455–464. IEEE, 2011.

[6] B. Dong, S. Byna, and K. Wu. Expediting scientific data analysis with reorganization. In *Proceedings of the IEEE Cluster 2013*, Cluster '13, 2013.

[7] B. Dong, X. Li, L. Xiao, and L. Ruan. A file assignment strategy for parallel i/o system with minimum i/o contention probability. In T.-h. Kim, H. Adeli, H.-s. Cho, O. Gervasi, S. Yau, B.-H. Kang, and J. Villalba, editors, *Grid and Distributed Computing*, volume 261 of *Communications in Computer and Information Science*, pages 445–454. Springer Berlin Heidelberg, 2011.

[8] L.-W. Lee, P. Scheuermann, and R. Vingralek. File assignment in parallel i/o systems with minimal variance of service time. *IEEE Trans. Comput.*, 49(2):127–140, Feb. 2000.

[9] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 328–336, Washington, DC, USA, 1994. IEEE Computer Society.

[10] X.-H. Sun, Y. Chen, and Y. Yin. Data layout optimization for petascale file systems. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 11–15, New York, NY, USA, 2009. ACM.

[11] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 93–102, Washington, DC, USA, 2011. IEEE Computer Society.

[12] K. Varda. Protocol buffers. http://code.google.com/apis/protocolbuffers/.

[13] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. G. R. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Laurent, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang. FastBit: Interactively Searching Massive Data. *Journal of Physics Conference Series, Proceedings of SciDAC 2009*, 180:012053, June 2009. LBNL-2164E.

[14] K. Wu, S. Byna, D. Rotem, and A. Shoshani. Scientific data services: a high-performance i/o system with array semantics. In *Proceedings of the first annual workshop on High performance computing meets databases*, HPCDB '11, pages 9–12, New York, NY, USA, 2011. ACM.

[15] T. Xie and Y. Sun. A file assignment strategy independent of workload characteristic assumptions. *Trans. Storage*, 5(3):10:1–10:24, Nov. 2009.

[16] Y. Zhu, Y. Yu, W. Y. Wang, S. S. Tan, and T. C. Low. A balanced allocation strategy for file assignment in parallel i/o systems. In *Proceedings of the 2010 IEEE Fifth International Conference on Networking, Architecture, and Storage*, NAS '10, pages 257–266, Washington, DC, USA, 2010. IEEE Computer Society.

[17] H. Zou, M. Slawinska, K. Schwan, M. Wolf, G. Eisenhauer, F. Zheng, J. Dayal, J. Logan, S. Klasky, T. Bode, M. Kinsey, and M. Clark. Flexquery: An online in-situ query system for interactive remote visual data exploration at large scale. In *Proceedings of the 2013 IEEE International Conference on Cluster Computing*, Cluster 2013, 2013.